Development of the PVFSv2 Parallel File System

Walt Ligon
ESTC
Wednesday, June 25, 2003

Relevance to NASA's Mission

- “Petabytes to Megabytes”
  - Dealing with Petabytes is a tough problem
  - Distillation ... not the final product
- Exaflops for modelling ...
  - How many bytes per second to deal with Exaflops?
- More and more and MORE data!
  - Even if we filter it down, there is a LOT to work with

My Work is About ...

- High Performance Computing
  - Computing when speed is really the issue
  - How to deliver Petaflops and Exaflops
- Parallel Computing
  - It is really the only way to get the speed we need
  - It fundamentally changes everything about how programs run
- Parallel I/O
  - If you can't keep it fed, it doesn't matter how fast it is

Parallel File Systems

- Two tasks:
  - Distribute data among nodes in a parallel computer
    - Leverage parallel I/O subsystems for performance
  - Provide access mechanisms for parallel applications
    - Simplifies complex access procedures
- PVFS – Parallel Virtual File System
  - Designed for Beowulf class parallel computers
  - Focuses on high-throughput access to large data sets
  - Underdevelopment since 1993 at GSFC

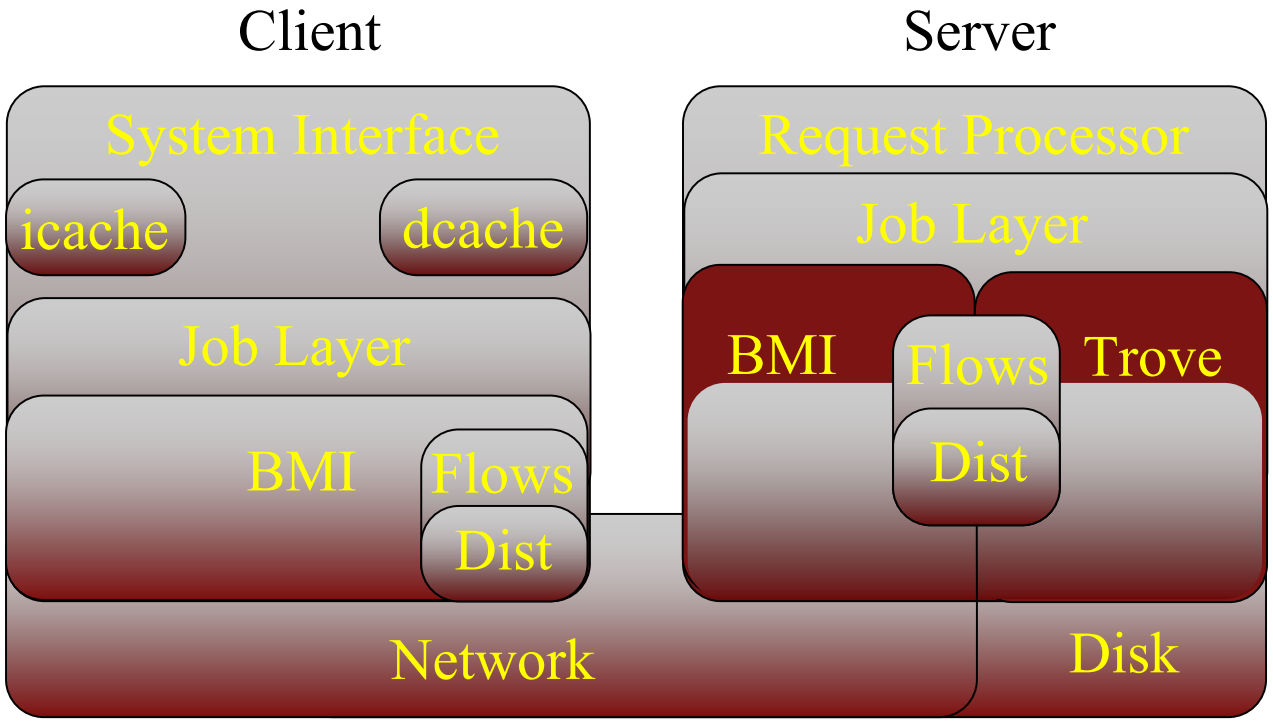
Goals and Outline

- Goals of this talk:
  - Briefly cover the design of PVFSv2
    - Goals of PVFSv2
    - Major subsystems
  - Motivate innovative features
    - Networking and storage abstractions
    - Requests and distributions
  - Introduce research directions
    - Redundancy
    - Semantics
    - Synchronization and atomicity

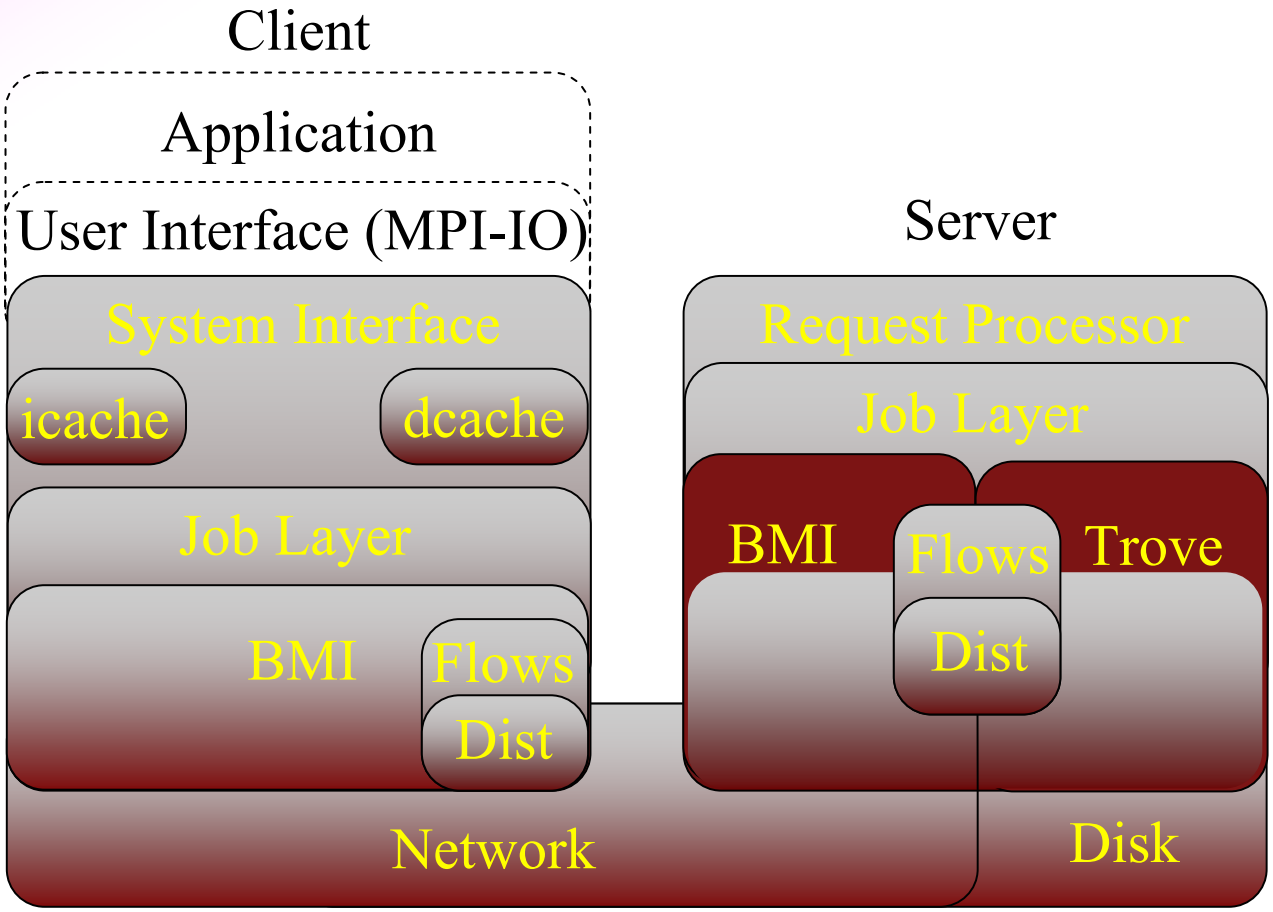
Goals of PVFSv2

- Production quality design and implementation
- Extendable with modules
- Fully distributed - data and metadata
- Support for MPI-IO
- Support for data redundancy and fail-over
- Support for experimentation and research in parallel file systems

PVFSv2 System Architecture



PVFSv2 System Architecture



BMI

- Network interface abstraction
  - Abstract node addressing
  - Supports multiple network fabrics simultaneously
  - Non-blocking semantics
    - post: submit a send or receive
    - test: check if a send or receive is done
    - wait: wait for a send or receive to be done
  - Ports to new network fabrics with module
  - Support for zero-copy protocols

Trove

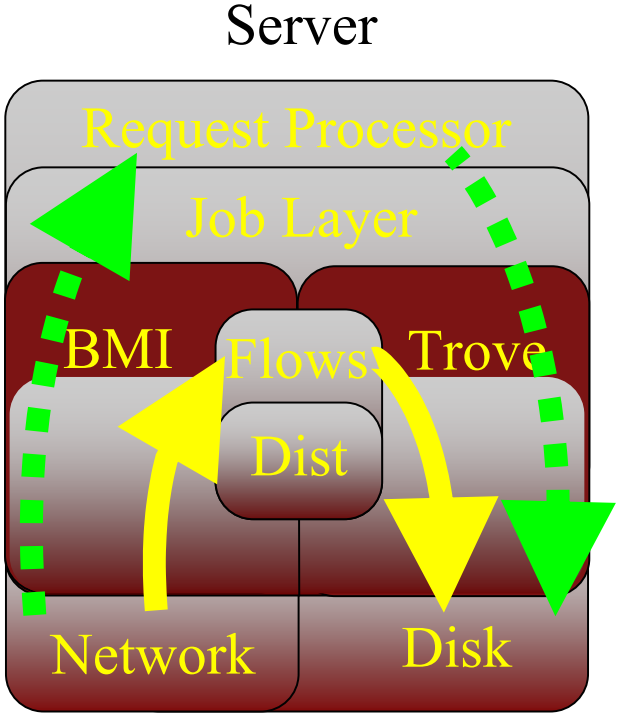
- Storage interface abstraction
  - Manages storage objects referenced by a handle
  - Non-blocking interface
  - Data space (storage object) has:
    - bytestream space -- used for file data
    - key/value pair space -- used for file metadata
    - attribute space -- used for object metadata (stat info)
  - Data spaces are clustered to allow migration of storage objects between servers
  - Support for synchronizing access to storage objects

Job Layer

- Manages asynchrony and scheduling for I/O transfers
  - Non-blocking interface
  - Combines operations for BMI, Trove, and Flows
    - Allows testing and waiting on all subsystems
    - May utilize threads for better performance
  - Performance dependency analysis on server requests and enforces ordering
  - Supports higher order scheduling between transfers

Flows

- Provide a direct transfer of data between:
  - BMI and Trove
  - BMI and memory
- Allows transfers to bypass job and request layers for efficiency
- Implements request and distribution processing



Request Processor

- Main control loop of the PVFSv2 server
- Designed to process multiple requests concurrently
- Implemented using a state machine that encodes the steps of each request
- State machine language implemented to simplify state machine coding
  - Allows new requests to be easily added
  - encourages code reuse

System Interface

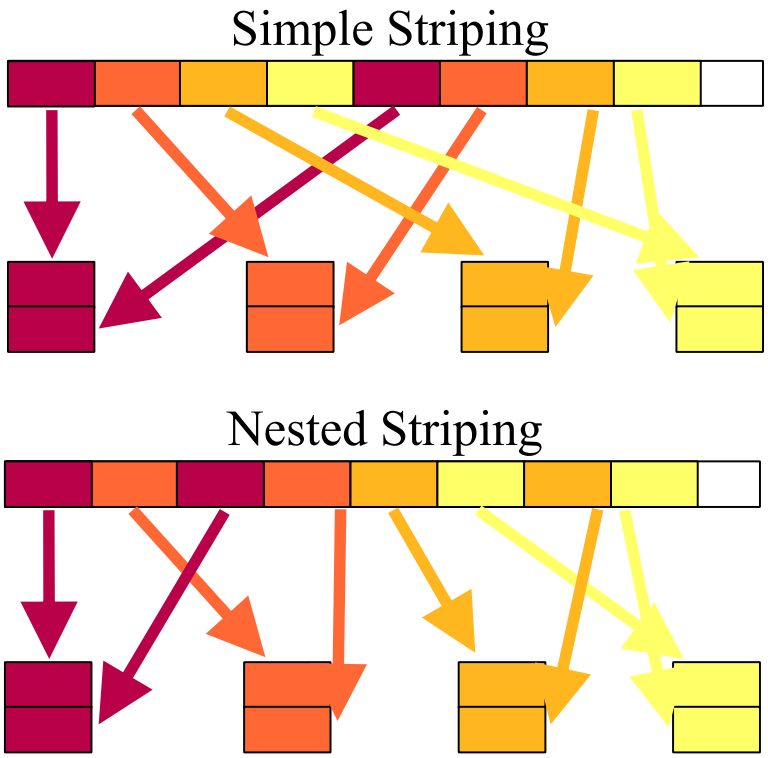
- Client-side top-level system code
- Implements caching of inodes and directories
- Exposes features of the file system for use in higher level interfaces
- Intended as a VFS-level interface, not a user interface

I/O Requests

- Based on MPI Datatype
  - Uses same constructor functions
  - Easily integrates with existing MPI Datatypes
  - Allows arbitrary non-contiguous regions to be read from a file
  - Provides a compact description of large regular access patterns

Data Distribution

- Programmable distributions are loaded from modules
  - Arbitrary mapping of data to servers
  - Data can be distributed to match algorithmic access patterns
  - Distribution can be fine tuned to maximize performance



Redundancy

- Redundancy is often at odds with performance
  - RAID 5 is considered the standard for redundancy
  - RAID 5 creates a bottleneck that requires ALL file data to pass through one point in the system to compute parity
  - RAID 5 is not generally the best performing configuration - mirroring (RAID 0+1) is usually better
  - Both RAID 5 and RAID 0+1 require synchronization between data servers for consistency

Redundancy in PVFSv2

- PVFSv2 will support redundancy
  - Selectable type of redundancy on a file-by-file basis
  - Redundancy implemented in modules - allows different redundancy schemes to be implemented
  - Fault-tolerant features in the system interface
    - tolerates server failure
    - reports server status
    - allows restart of requests

New Redundancy Types

- Configure when redundancy is maintained
  - Lazy Redundancy
    - Redundancy created when file closed
    - Protects against loss during storage
  - Commit Redundancy
    - Redundancy created at specific points in program execution
    - Useful for long running programs that update by section
  - Update Redundancy
    - Redundancy updated when data updated
    - Most complex, and most secure

Semantics

- Portability concerns
  - Posix
  - MPI-IO
- Caching
- Locking
- Concurrent access
- Security

PVFSv2 Semantics

- Guiding principles
  - Semantics often conflict with performance goals
  - No single set of semantics is right for every situation
- High-performance choices
- Implementations of alternative choices supported
  - caching
  - redundancy
  - locking
- Expect more choices in the future

Synchronization and Atomicity

- Needed by
  - Redundancy schemes
  - Some semantics
  - Some applications
- Mostly implemented with region-based locks
  - Work well in hardware but not scalable in software
  - Mostly used to achieve atomicity
  - Lots of state on clients
  - Lots of I/O, poor scalability

Conditional Operators

- Taken from modern SMP hardware designs
  - Load Locked
  - Store conditional
- Allows local operations to proceed
- Conditional store operations check for atomicity violation
- Could this be applied to a parallel file system?

Benchmarking

- Need standardized benchmarks for parallel I/O
  - measurement procedure
  - reporting format
  - terminology
- Test a range of workloads
  - small/large transactions
  - contiguous/non-contiguous
  - metadata operations
- Both synthetic and application benchmarks

I/O Benchmark Consortium

- Open group working to establish an effective set of benchmarks for parallel I/O
- Have national lab and university involvement
- Need industry involvement
- Need input from applications groups

`http://www.mcs.anl.gov/~rross/pio-benchmark/index.html`

Conclusions

- PVFSv2 development nearing a release
  - Targeting Supercomputing 2003 Exposition
- Important research issues
  - Locking, redundancy, scalability
  - Interfaces, semantics
- We need a joint effort to reach goals
  - Open, flexible, common platform
  - Good benchmarks

`http://www.parl.clemson.edu/pvfs/`